SQL wrap-up; NULLs
Relational Algebra intro

Last time

- Basic SQL features
  - AND/OR, UNION/INTERSECT/EXCEPT
- Advanced SQL features
  - Nested queries and correlation
  - Aggregates (MAX, COUNT)
  - GROUP BY and HAVING

Null Values

- Represents values that are unknown (e.g., grade has not been assigned) or inapplicable (e.g., no SSN for a non-US-resident)
  - SQL provides a special value NULL
  - Can specify a field cannot be null using NOT NULL in definition

sid	sname	ssn
11	alice	123
22	bob	NULL

Null Values

- Semantics of operators must be designed carefully
  - 3-valued logic (true, false, unknown)

```
SELECT 3 = NULL;
SELECT NULL = NULL;
SELECT NULL IS NULL;
SELECT NULL IS NOT NULL;
SELECT TRUE OR NULL;
SELECT TRUE AND NULL;
```

```
SELECT * FROM Sailors2 S WHERE S.ssn IS NULL;
```

```
SELECT * FROM Sailors2 S WHERE S.ssn > 124;
SELECT * FROM Sailors2 S WHERE S.ssn <= 124;
```

Outer Joins

- Suppose we join Sailors and Reserves, but some sailors have not made a reservation
- Can imagine wanting to retain all Sailors tuples in the result
- Can do this with an OUTER JOIN

```
SELECT * FROM SAILORS LEFT OUTER JOIN RESERVES USING (sid);
```

- Also available: right outer join, full outer join
  - What do you think they do?

Outer Join Example

- For each sailor, display their id, name and how many reservations they have made in total
  - Including 0 if no reservations;

Outer Join Example

```
SELECT sid, sname, count(bid)
FROM (SELECT * FROM SAILORS JOIN
          RESERVES USING (sid)) AS TEMP
GROUP BY sid, sname;
```

```
SELECT sid, sname, count(bid)
FROM (SELECT * FROM SAILORS LEFT OUTER JOIN
          RESERVES USING (sid)) AS TEMP
GROUP BY sid, sname;
```

Be careful with NULLs!

```
SELECT sid, sname, count(bid)
FROM (SELECT * FROM SAILORS LEFT OUTER JOIN
        RESERVES USING (sid)) AS TEMP
GROUP BY sid, sname;
```

```
SELECT sid, sname, count(*)
FROM (SELECT * FROM SAILORS LEFT OUTER JOIN
        RESERVES USING (sid)) AS TEMP
GROUP BY sid, sname;
```

SQL Summary:

- Basic SELECT/FROM/WHERE queries
- Expressions and strings
- Set operators
- Nested queries
- Aggregation
- GROUP BY/HAVING
- Null values and Outer Joins
- (ORDER BY and other features…)

How to write complex SQL queries?

- Gets easier with practice
- Look at lots of examples (eg in your textbook)
  - Including the textbook exercises
  - Solutions to odd-numbered exercises freely available on textbook's website http://pages.cs.wisc.edu/~dbbook/

How to write complex SQL queries?

- Bottom-up strategy
  - Can I write anything that is even slightly related to the query I need?
  - Maybe you can use that as a subquery/building block
- Top-down strategy
  - Suppose I could ask for a "magic table" given to me by someone smart, and use that as a subquery
  - What would that table be and how would I use it?

How to write complex SQL queries?

- Can you compute the opposite (negation/ complement) of what you want
- Sailors who have reserved all boats?
  - How about the sailors who have NOT reserved all boats?

How to write complex SQL queries?

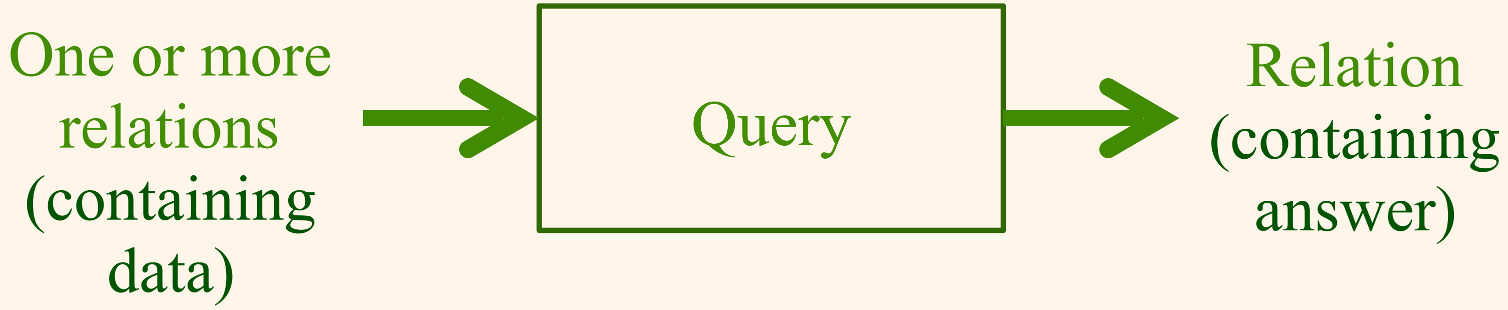
- Consider "instantiating" something in your query to make it simpler
- E.g. "find sailors who have reserved all boats"
  - If that's too hard, can you do "find out whether sailor with sid 1 has reserved all boats"?
  - I.e. write a query whose answer is empty if no, nonempty if yes (or vice versa)
- E.g. a GROUP BY query
  - If you don't know how to do something for every pair of sailorids, can you do it for a concrete pair, e.g. (1, 2)?

Query Evaluation

- Your SQL query is parsed and translated into an intermediate representation
  - Based on relational algebra
- This allows the query to be optimized
  - May be several possible ways to compute the answer
  - And the best strategy may depend on features of the actual data
  - The DBMS is clever; can evaluate options and choose the best one

Relational algebra

- A simple, powerful abstraction for representing SQL queries
- Based on the idea of operators

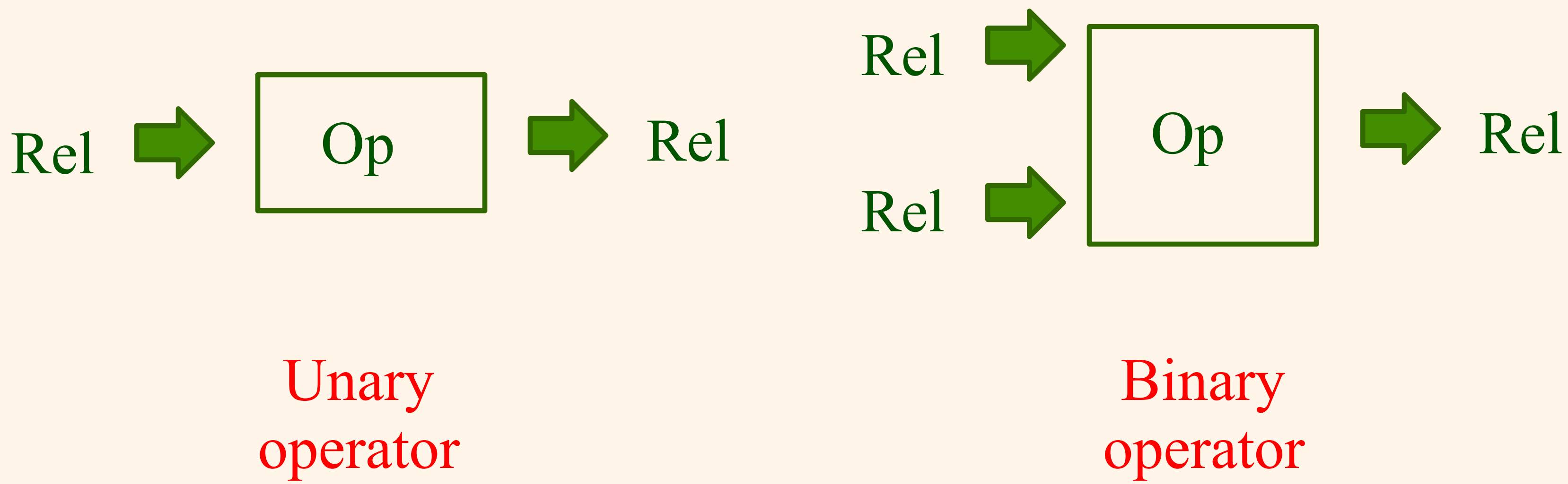


Preliminaries

- A query is applied to relation instances, and the result of a query is also a relation instance.
  - So, operators are composable
- Start with set relational algebra – inputs and outputs are *sets* of tuples, no duplicates

Operators

- A relational algebra operator takes as input one or more relations and outputs another relation



The unary RA operators

- Select and Project

Selection operator

- Input: a relation
- Output: a relation containing a <u>subset of the tuples</u> from the input relation
  - That satisfy a certain condition

bid	name	color
101	Misty	red
102	Pearl	blue
103	Speedy	blue

selection

bid = 101

bid	name	color
101	Misty	red

Projection operator

- Input: a relation
- Output: a relation containing a subset of the columns from the input relation

bid	name	color
101	Misty	red
102	Pearl	blue
103	Speedy	blue

color
red
blue

- Relations are sets, so duplicates removed!!

We can already use these

- Suppose you didn't know SQL, but knew selection and projection
  - Query: find the color of boat number 101

The color of boat 101

bid	name	color
101	Misty	red
102	Pearl	blue
103	Speedy	blue

selection

bid = 101

bid	name	color
101	Misty	red

projection

color

color
red

The color of boat 101

projection

```
SELECT B.color
FROM Boats B
WHERE B.bid= '101';
```

selection

In more formal notation

- Special symbols for operators
- Selection: σ
- Projection: π
- Our example query:

$$\pi_{color}(\sigma_{bid=101}(Boats))$$

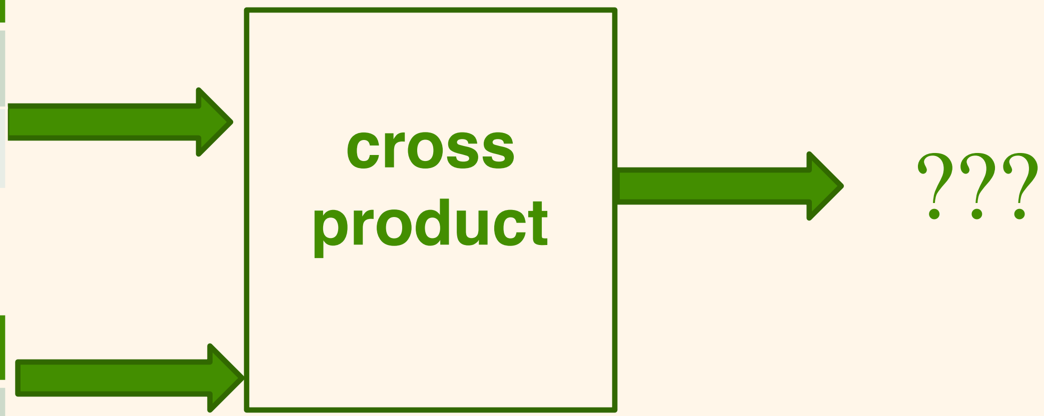
Cross Product

- Input: two relations
- Output: a relation containing all pairs of tuples from both relations

Cross Product

bid	name	color
101	Misty	red
102	Pearl	blue

sid	bid	day
22	101	9/10/13
58	102	9/11/13



From SQL to RA

- Projection, selection, cross product allow us to express most SELECT-FROM-WHERE queries
- Let’s try it on our running example:

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid='101';
```

In mathematical notation

```
SELECT    S.sname
FROM      Sailors S, Reserves R
WHERE     S.sid=R.sid AND R.bid='101';
```

$$\pi_{S.sname}(\sigma_{R.bid=101 \wedge R.sid=S.sid}(R \times S))$$